

# 期中复习

---

苏州大学 计算机科学与技术学院

汪笑宇

Email: [xywang21@suda.edu.cn](mailto:xywang21@suda.edu.cn)

# 基本概念——算法

## ■非形式定义

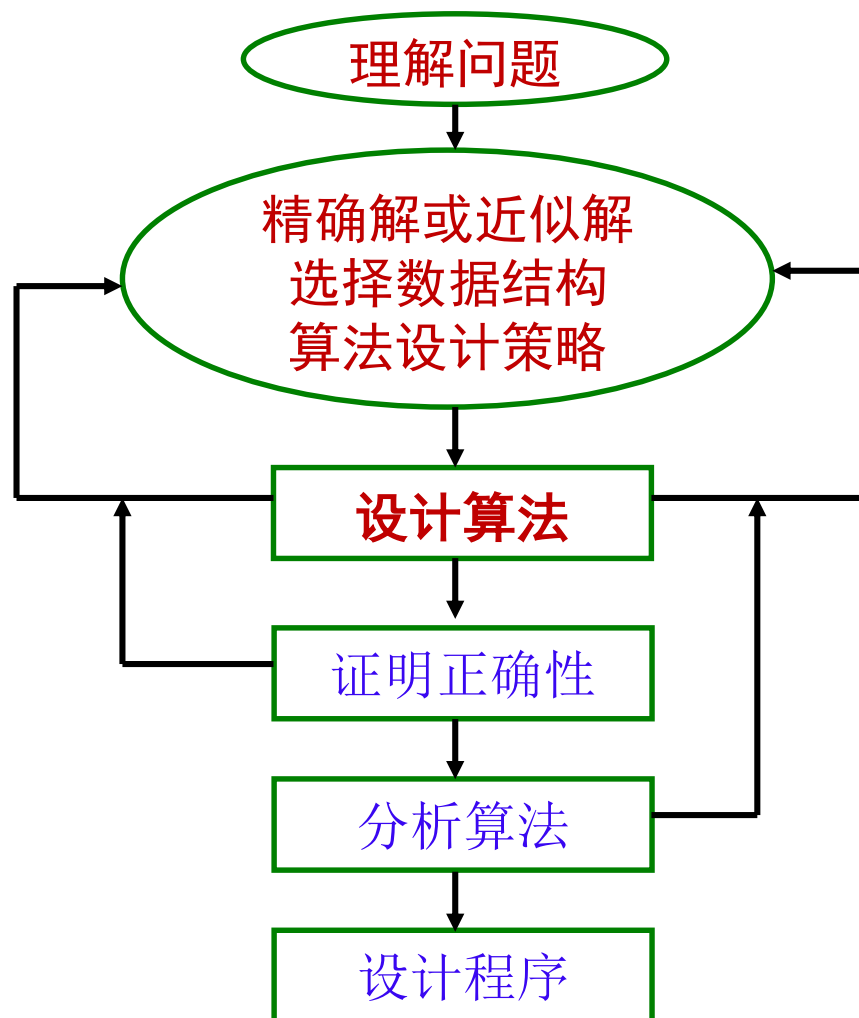
- 一个**算法**是任何一个**良定义(well-defined)**的计算过程，它接收某个值或值的集合作为输入，产生某个值或值的集合作为输出。因此，一个算法是一个**计算步骤的序列**，这些步骤将输入转化为输出。



- 或者说，算法所描述的计算过程就是**怎样达到所期望的I/O关系**

# 算法 -> 问题求解

---



# 循环不变式

---

■循环不变式：循环体每次执行前后均为真的谓词

➤作用：主要用来帮助我们理解算法的正确性。

■循环不变式性质：

➤初始化：循环的第一次迭代之前，它为真

➤保持：如果循环的某次迭代之前它为真，那么下次迭代之前它仍为真

➤终止：在循环终止时，不变式为我们提供一个有用的性质，该性质有助于证明算法是正确的

# 渐近表示法

## ■ $f(n)=\Theta(g(n))$ 渐近紧确界

$\Theta(g(n)) = \{f(n): \text{存在正常量 } c_1, c_2, n_0, \text{ 使得对所有 } n \geq n_0 \text{ 有 } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$

## ■ $f(n)=O(g(n))$ 渐近上界

$O(g(n)) = \{f(n): \text{存在正常量 } c, n_0, \text{ 使得对所有 } n \geq n_0 \text{ 有 } 0 \leq f(n) \leq c g(n)\}$

## ■ $f(n)=\Omega(g(n))$ 渐近下界

$\Omega(g(n)) = \{f(n): \text{存在正常量 } c, n_0, \text{ 使得对所有 } n \geq n_0 \text{ 有 } 0 \leq c g(n) \leq f(n)\}$

# 渐近表示法 (续)

## ■ $f(n)=o(g(n))$ 渐近非紧上界

$o(g(n)) = \{f(n): \text{对任意常数 } c>0, \text{ 存在常数 } n_0 > 0, \text{ 使得对所有 } n \geq n_0 \text{ 有 } 0 \leq f(n) < cg(n)\}$

## ■ $f(n)=\omega(g(n))$ 渐近非紧下界

$\omega(g(n)) = \{f(n): \text{对任意常数 } c>0, \text{ 存在常数 } n_0 > 0, \text{ 使得对所有 } n \geq n_0 \text{ 有 } 0 \leq cg(n) < f(n)\}$

$$\blacksquare \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & f(n) = o(g(n)), f(n) = O(g(n)) \\ a, a > 0 & f(n) = \Theta(g(n)), f(n) = O(g(n)), f(n) = \Omega(g(n)) \\ \infty & f(n) = \omega(g(n)), f(n) = \Omega(g(n)) \end{cases}$$

# 标准记号与常用函数

---

## ■阶乘

### ➤斯特林(Stirling)近似公式

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

$$\text{➤ } n! = o(n^n) \quad n! = \omega(2^n) \quad \lg(n!) = \Theta(n \lg n)$$

# P类问题 (P问题)

---

■P类问题 (Polynomial Problem): 在多项式时间内可以解决的问题

➤例如: 排序问题可在 $O(n^2)$ 时间复杂度内解决

# NP类问题 (NP问题)

---

## ■NP类问题 (Non-deterministic Polynomial Problem): 在多项式时间内可以被验证的问题

- 非确定性算法：猜测 + 验证，猜测阶段是非确定性的，给出问题解的一个猜测；验证阶段是确定性的，验证阶段给出解的正确性
- 设算法A是解一个判定问题Q的非确定性算法，如果算法A的验证阶段可以在多项式时间内完成，则称算法A是一个多项式时间非确定性算法，也称问题Q是非确定性多项式时间可解的（NP问题）

# NPC类问题 (NPC问题)

---

■若对于问题Q，满足：

1.  $Q \in \text{NP}$

2. 每个NP问题都可以多项式时间归约到Q

或：任意一个NPC问题都可以多项式时间归约到Q

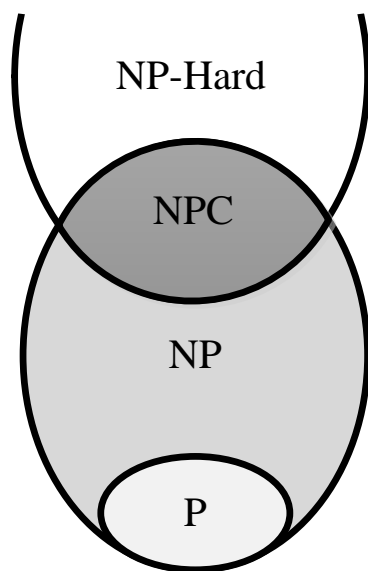
那么问题Q是一个NP完全问题

■若问题Q满足了第二条，则称为NP-Hard问题

# NP完全性理论

■定理34.4 (p628) 如果任何NP完全问题是多项式时间可求解的，则 $P=NP$ 。等价地，如果存在某一NP中的问题不是多项式时间可求解的，则所有NP完全问题都不是多项式时间可求解的。

■目前认为 $P \neq NP$



# 递归式求解——代入法

---

■（教材p47-49）代入法求解递归式分为两步：

➤猜测解的形式

➤用数学归纳法求出解中的常数，并证明解是正确的

■关键：用猜测的解代入到递归式中

■例：确定  $T(n) = 2T(\lfloor n/2 \rfloor) + n$  的上界

➤猜测解  $T(n) = O(n \lg n)$ ，需要证明恰当选择常数  $c > 0$ ，有  $T(n) \leq cn \lg n$

# 递归式求解——代入法 (续)

■例：确定 $T(n) = 2T(\lfloor n/2 \rfloor) + n$ 的上界

- 猜测解 $T(n)=O(n\lg n)$ ，需要证明恰当选择常数 $c>0$ ，有 $T(n)\leq cn\lg n$
- 先看此猜测是否正确：假定该上界对所有正数 $m<n$ 都成立，特别是对于 $m = \lfloor n/2 \rfloor$ 有

$$T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)$$

代入递归式得到：

$$\begin{aligned} T(n) &\leq 2c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor) + n \leq cn \lg(n/2) + n \\ &= cn \lg n - cn \lg 2 + n \\ &= cn \lg n + (1 - c)n \leq cn \lg n \end{aligned}$$

只要 $c\geq 1$ 即可

# 递归式求解——代入法 (续)

■例：确定 $T(n) = 2T(\lfloor n/2 \rfloor) + n$ 的上界

- 猜测解 $T(n)=O(n\lg n)$ ，需要证明恰当选择常数 $c>0$ ，有 $T(n)\leq cn\lg n$
- 数学归纳法要求证明边界条件成立：假设 $T(1)=1$ 是递归式唯一边界条件，但 $T(1)\leq c\cdot 1\cdot \lg 1=0$ 并不成立
- 渐近符号仅要求对 $n\geq n_0$ 证明 $T(n)\leq cn\lg n$ 
  - 扩展边界条件
  - 令 $n_0=2$ ，之前的证明成立需要对 $n_0 \leq m = \lfloor n/2 \rfloor < n$ 都成立，因此数学归纳法中需满足 $n\geq 4$ ，此时只要找到足够大的 $c$ 使得 $T(2)$ 和 $T(3)$ 满足 $T(n)\leq cn\lg n$ 即可
  - $T(2)=4$ ， $T(3)=5$ ，即任何 $c\geq 2$ 即可满足

# 递归式求解——代入法 (续)

---

## ■ 注意事项3：避免陷阱

➤ 证明时渐近记号的使用易产生错误

例：  $T(n) = 2T(\lfloor n/2 \rfloor) + n$

猜测  $T(n) = O(n)$ ，此时“证明”  $T(n) \leq cn$

$T(n) \leq 2(c\lfloor n/2 \rfloor) + n \leq cn + n = O(n)$  错误！

必须证明  $T(n) \leq cn$  的精确形式！

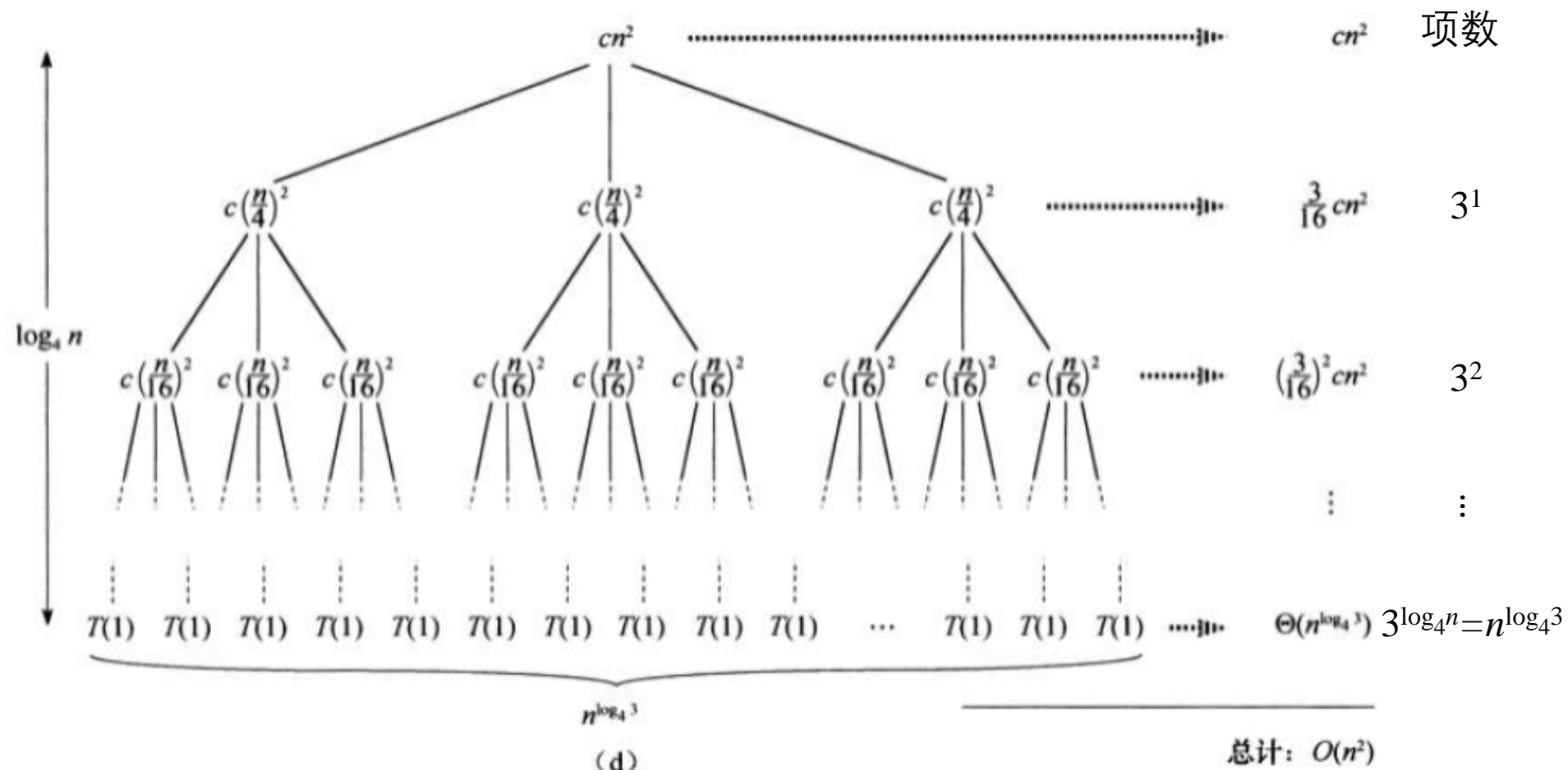
# 递归式求解——递归树法

---

- 递归树中每个结点表示一个单一子问题的代价，子问题对应某次递归函数调用
- 树中每层的代价求和得到每层代价，将所有层的代价求和，得到所有层次递归调用总代价
- 递归树是展开过程的形象化，从 $T(n)$ 逐步展开直到 $T(1)$

# 递归式求解——递归树法 (续)

■例2: (教材p51)  $T(n) = 3T(\lfloor n/4 \rfloor) + cn^2$  (设  $n=4^k$ )



# 递归式求解——递归树法 (续)

■例2: (教材p51)  $T(n) = 3T(\lfloor n/4 \rfloor) + cn^2$  (设  $n=4^k$ )

$$\begin{aligned} T(n) &= cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \dots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3}) \\ &= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\ &= \frac{1 - (3/16)^{\log_4 n}}{13/16} cn^2 + \Theta(n^{\log_4 3}) \\ &< \frac{16}{13} cn^2 + \Theta(n^{\log_4 3}) \quad // 0 < (3/16)^{\log_4 n} \leq 1 \\ &= O(n^2) \end{aligned}$$

# 递归式求解——主方法

---

■定理4.1（教材p53-54，主定理） 令 $a \geq 1$ 和 $b > 1$ 是常数， $f(n)$ 是一个函数， $T(n)$ 是定义在非负整数上的递归式：

$$T(n) = aT(n/b) + f(n)$$

其中我们将 $n/b$ 解释为 $\lceil n/b \rceil$ 或 $\lfloor n/b \rfloor$ ，有如下渐近界：

1. 若对某个常数 $\varepsilon > 0$ 有 $f(n) = O(n^{\log_b a - \varepsilon})$ ，则 $T(n) = \Theta(n^{\log_b a})$
2. 若 $f(n) = \Theta(n^{\log_b a})$ ，则 $T(n) = \Theta(n^{\log_b a} \lg n)$
3. 若对某个常数 $\varepsilon > 0$ 有 $f(n) = \Omega(n^{\log_b a + \varepsilon})$ ，且对某个常数 $c < 1$ 和所有足够大的 $n$ 有 $af(n/b) \leq cf(n)$ ，则 $T(n) = \Theta(f(n))$

# 递归式求解——主方法 (续)

---

■定理意义：比较 $f(n)$ 和 $n^{\log_b a}$ ，直观上两函数较大者决定 $T(n)$

1.  $n^{\log_b a}$ 比 $f(n)$ 大一个多项式因子 $n^\varepsilon$ ：  $T(n) = \Theta(n^{\log_b a})$

2. 两者相同，乘以对数因子 $\lg n$ ：

$$T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(\lg n f(n))$$

3.  $f(n)$ 比 $n^{\log_b a}$ 大一个多项式因子 $n^\varepsilon$ ，以及满足“正则”条件：  $T(n) = \Theta(f(n))$

■注：三种情况并未覆盖所有可能的 $f(n)$ ，存在间隙

# 分治法求解——二分搜索

```
BINARY_SEARCH(A, x, low, high)  
1  if low ≤ high  
2    mid ← ⌊(low + high)/2⌋  
3    if x = A[mid]  
4      return mid  
5    if x < A[mid]  
6      return BINARY_SEARCH(A, x, low, mid-1)  
7    else return BINARY_SEARCH(A, x, mid+1, high)  
8  return 0
```

■最坏情况运行时间：  $T(n)=T(n/2)+\Theta(1)$ ，由主方法易得  $T(n)=\Theta(\lg n)$